(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 737 922 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
16.10.1996 Bulletin 1996/42

(51) Int Cl.6: G06F 9/46

(21) Application number: 96301560.7

(22) Date of filing: 07.03.1996

(84) Designated Contracting States:
DE FR GB IT SE

(30) Priority: 22.03.1995 US 408645

(71) Applicant: SUN MICROSYSTEMS, INC.
Mountain View, California 94043-1100 (US)

(72) Inventors:
- Menges, Anthony W.
San Jose, CA 95120 (US)
- Cavanaugh III, Kenneth M.
Montara, CA 94037-1480 (US)
- Brownell, David M.
Palo Alto, CA 94306 (US)
- Hagmann, Robert B.
Palo Alto, CA 84303 (US)
- Hare, Dwight F.
La Selva Beach, CA 95076 (US)
- Vanderbilt, Peter
Mountain View, CA 94043 (US)

(74) Representative: Browne, Robin Forsythe, Dr.
Urquhart-Dykes & Lord
Tower House
Merrion Way
Leeds LS2 8PA West Yorkshire (GB)

(54) Method and apparatus for managing computer processes

(57) A number of methods, apparatus, and data structures are disclosed for managing computer processes. In one aspect, a daemon process which manages server processes includes an active server table and a locator service. The locator service can look up and register server processes in the active server table. Furthermore, the locator service can start up server processes. In some embodiments, the locator service includes a look-up object and a server process registration object which perform the tasks of the locator service. In other embodiments, methods for managing server process such as starting and registering the server processes are taught. In one specific method, a daemon process performs a variety of steps in response to receiving a look-up call for a target object. These steps include obtaining a server identifier for the target object, determining the state of a server process, and returning addressing information corresponding to the server process under which the target object will activate. In related method aspects the daemon process will start the server process if it isn't running and/or wait until the server process is running to return the addressing information. In a separate method aspect, a server process self-starts; receiving an object reference for a desired target object, receiving a server process identification number, creating a communications port for itself, forming addressing information for itself, obtaining an object reference for a server process registration object, and registering itself by calling the server process registration object to invoke a register new process operation.

EP 0 737 922 A1

an object adapter database resident therein. In additional embodiments, the object adapter database is resident elsewhere in the distributed object operating environment. One specific embodiment has the object adapter database in a separate process running on the host computer. The object adapter database includes data elements such as target object identifiers, server identifiers each corresponding to at least one target object identifier, and object references, each object reference corresponding to a target object identifier. In further embodiments, the locator service includes two objects, a locator object and a server process registration object. The locator object performs the server look-up function, and the server process registration object performs the server process registration function. A still further embodiment is contemplated wherein a distributed object operating environment includes a multiplicity of computer systems connected by a network with a distinct daemon process as described above running on each one of the computer systems.

In a separate aspect of the present invention, a computer system for use in a distributed object operating environment is contemplated. The computer system includes a central processing unit, memory accessed by the central processing unit, and a locator service implemented on the computer system. A plurality of distributed objects reside in the memory. The locator service manages the distributed objects and any server processes executing on the computer system. In several related embodiments, the computer system will have means for performing the method aspects of the invention as described below.

In other aspects of the invention various methods for managing server processes such as starting and registering server processes are disclosed. These server processes reside on a server host computer. In a first method, a daemon process resident on the server host computer performs the steps of receiving a look-up call for a target object, obtaining a server identifier for said target object, determining a state of a server process, and returning addressing information corresponding to said server process. The server process is uniquely defined within the server host computer by way of the server identifier. In one method aspect, the server process state is determined to be running and, as a result, the step of returning said addressing information is performed immediately. In another method aspect, the state is determined to be starting and, as a result, the daemon process waits until the state transitions from starting to running before returning the addressing information.

In another method aspect, the step of obtaining a server identifier for the target object is performed by accessing a first database. By way of example, the first database may be an object adapter database. Furthermore, the state of the server process is determined by looking it up in a second database. In one example, the second database may be an active server table. In another embodiment, the first and the second databases are the same database.

In a still further method aspect, when the state of the process is determined to be not active (*i.e.* the server process is not found in the second database), the daemon process creates a server process entry in the second database. This entry includes a server identifier and a server process state. The daemon process then continues by marking the server process state as starting, accessing the first database to get an execdef for the target object, starting the server process using the execdef, and waiting until the server process state transitions from starting to running before returning the server process addressing information to the client. By way of example, the execdef may be the server's program name and any necessary arguments.

In yet another method aspect, the server process responds to being started by obtaining a server process identification number from the operating system of the host computer. Then it creates a communications port for the server process, forms addressing information for the server process, and then calls a server process registration object resident in the daemon process. The operation of this call will invoke a register server operation. As arguments to this call, the server process passes the addressing information, the server process identification number, and the server identifier. In response to this call, the server process registration object stores the addressing information in the second database and marks the server process state entry as running.

In another separate aspect of the invention, a self-start method for a server process (*i.e.* self-start and register) is taught. The method begins when the server process receives a request to become a server along with an object reference for the desired target object. In response the server process begins executing and obtains a server process identification number from the operating system of the host computer. Next it performs the steps of creating a communications port for the server process, forming addressing information for the server process, obtaining an object reference for a server process registration object from an object request broker object file and calling the server process registration object to invoke a register new server operation. Then, in response to this call, the server process receives a server identifier corresponding to itself.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention, together with further objectives and advantages thereof, may best be understood by reference to the following description taken in conjunction with the accompanying drawings in which:

FIGURE 1 is a pictorial illustration of various computers linked together in a computer network.

FIGURE 2 illustrates diagramatically the major

of the objects.

A "distributed object system" or "distributed object operating environment" refers to a system comprising distributed objects that communicate through an ORB.

An "object reference" or "objref" is an object that contains a pointer to another object. For example, the objref may contain addressing information such as a host computer network address, an ORB daemon network port address and an object identifier. The creation and definition of object references will be familiar to those skilled in the art.

A "client" as defined herein refers to an entity that sends a request to an object. In this model, the object receiving the request is referred to as a "server object" or a "target object". Thus, clients invoke operations, or implementations, from servers. In a distributed object environment, clients need not have knowledge of the implementation programming language, nor does the implementation have to have knowledge of the client's programming language due to the requirement of multilingual character of such objects. Clients and servers in distributed object environments need only communicate in terms of the interface definition language. As noted above, the request by the client to the server, and the server's reply to the client, is handled by the ORB. It should be pointed out that the client and server can exist within the same process, on the same host computer, or on two different host computers.

An "object interface" is a specification of the operations, attributes, and exceptions that an object provides. Preferably, object interfaces for distributed objects are written using IDL. As noted above, objects perform transactions through their interfaces. The use of interfaces therefore relieves the need of objects that are aware of the programming languages used to define the methods and data of the objects in the transaction.

To "marshal" a packet of information is to prepare this information for transfer over a network communications line. This often means organizing the data in a particular format in accordance with the network communications protocol being used.

To "unmarshal" a packet of information is to essentially reverse the marshaling procedure and produce data in a format which is meaningful in a non-network environment.

II. Managing Distributed Objects and Computer Processes

In a distributed object operating environment, requests and replies are made through an Object Request Broker (ORB) that is aware of the locations and status of the objects. One architecture which is suitable for implementing such an ORB is provided by the Common Object Request Broker Architecture (CORBA) specification. The CORBA specification was developed by the Object Management Group (OMG) to define the distributed computing environment world in terms of objects in a distributed client-server environment, where target objects are capable of providing services to clients requesting the service. In the following discussion, the terms "object" and "distributed object" will be used interchangeably, as the following invention is directed to both types.

According to the present invention, the ORB is responsible for managing many aspects of the distributed objects and the server processes found within the distributed object operating environment, including client-server interactions which involve calls to distributed objects. Distributed objects, as contemplated by the present invention, are implemented (by the ORB and/or the host computer) under computer processes. As is well known to those skilled in the art, computer processes provide a common framework under which computer systems function.

In actuality, a process typically includes an address space (*i.e.* a portion of memory allocated to only the process), a set of file descriptors, a process identification number, and one or more threads of execution (often referred to as threads). As is familiar to those skilled in the art, a single thread of execution is essentially a sequential flow of the point of execution through a process. Multi-threaded systems, such as the present invention, allow for multiple threads to run concurrently in a single process. For a more detailed description of threads, multi-threaded processes, and principles of concurrency, please see "Concurrency Within DOE Object Implementations" by Dr. Robert Hagmann, Version 0.91, May 27, 1993, published by SunSoft and incorporated herein by reference in its entirety.

As a direct result of the framework of computer processes, all entities residing under a single process will share resources (*i.e.* memory and files). Thus multiple target objects residing in a single process will have efficient communication with one another. Furthermore, data can be loaded into memory that all objects residing in the single process will have access to. However, programmers may have other motivations (beyond efficient transport and data sharing) which negate the advantages gained by having many objects in a single process. For instance, different objects will have different objectives and may rely on different assumptions about the process. These motivations generate a need for orderly, multi-process distributed object operating environments as disclosed by the present invention. In allowing programmers to keep objects within separate processes, the ORB may prevent conflict between and maintain the integrity of objects within processes. As a case in point, an object in a first server process may go into an error condition and begin chaotically writing within its server process memory. Nevertheless, objects running in separate server processes will remain intact since these processes have their own memory, files, and flow control.

In a preferred embodiment of the present invention, distributed objects and computer processes are resi-

directs the client entity to a locator service which can direct the client entity to the object. By way of a descriptive analogy, if a client requested geographical directions, an indirection would point the client to the location of a current map, or perhaps provide the client with a phone number of a geographically astute individual. The idea of using an indirection (rather than direct addressing) is useful for distributed objects as certain distributed objects (those of a persistent kind) may be mobile both from process to process and host computer to host computer.

Similar to the client object 302, the target object 304 of Fig. 3 may reside in a server process 318 which is running on a server host 320. Other entities may reside on both the server host 320 and the client host 308. These entities include but are not limited to additional processes and/or objects running under the client process, the server process, and/or the additional processes. Two suitable processes are daemon_1 process 317 running on client host 308 and daemon_2 process 321 running on server host 320. Daemon processes typically run in the background and are well known to those skilled in the art.

Fig. 4 shows a "process-to-process" client-server model in which the client object 302 and the target object 304 share the same host computer, the client/server host 322. However, client object 302 and target object 304 reside within different processes. Similar to Fig. 3, the client object 302 exists in a client process 306 which is running on the client/server host 322. The client object 302 includes a surrogate object 310 which provides an indirection 312 to an object reference 314 which in turn provides an indirection 316 to the target object 304. Target object 304 exists within in a server process 318 which is running on the client/server host 322. Akin to the "host-to-host" case, further entities such as processes and/or objects may be resident on client/server host 322. One such suitable process is daemon_1 process 317.

Fig. 5 shows a client-server model in which both client object 302 and target object 304 reside within the same process, the client/server process 324. Once again, the client object 302 uses a first surrogate object 310 which holds (*e.g.* it has a first indirection 312 to) an object reference 314. In turn, object reference 314 provides a second indirection 316 to the target object 304. In this case the second indirection 316 may just be a pointer to shared memory. This situation is perhaps the best of all three described cases. No network communications is necessary, and since both objects are under one process, they coexist in memory allocated to the client/server process 324. In the situation of Fig. 5, other entities such as objects may be resident in client/server process 324. For example, there may be other surrogate objects present in the client process 306, such as a second surrogate object 311, which may also hold the object reference 314. However some distributed object operating environments do not include surrogate objects. In these cases, the client 302 will utilize the object reference 314 without a surrogate object.

Two other typical client-server scenarios which are not shown in the above described Figs. will now be discussed briefly. The first set of scenarios involve the instances wherein the client is not an object. As will be appreciated by those skilled in the art, the issues which arise are very similar whether the client is an object or some other entity requesting service. The other scenario is one wherein the client object and the target object are the self-same object. This may occur when an object makes a recursive call upon itself. While the recursive call may appear unusual, this client-server interaction is a fairly common and powerful tool and may be dealt with in a manner similar to the case when a client object and a target object are unique but exist in the same process.

Fig. 6 is a pictorial illustration of a client-server interaction wherein an ORB daemon process manages a multiplicity of server process and target objects. Fig. 6 provides a generic paradigm for use in several specific embodiments of the present invention. The paradigm of Fig. 6 includes a client 350, a server process 352, and an ORB daemon process 354. In a step 356, client 350 calls ORB daemon process 354 to request addressing information for a target object. Then, if necessary, ORB daemon process 354 starts server process 352 in a step 358. Server process 352 corresponds to the server process for the requested target object. In a step 360, server process 352 responds to step 358 by indicating to ORB daemon process 354 that it is ready to provide services. In a step 362, ORB daemon process 354 returns addressing information to client 350. Once server process 352 is ready and client 350 has the addressing information, client 350 can call server 352 in a step 364 and server 352 can pass, in a step 366, the results of the call of step 364.

The paradigm of Fig. 6 provides a general overview of the client-server interaction in accordance with the present invention. The teaching of this invention is generally directed to methods, apparatus, and data structures which provide mechanisms for steps 358, 360, and 362. Other steps are described in more detail in Vanderbilt et. al.'s copending U.S. Patent Application Serial Number (Attorney Docket No. P717/SUN1P023) entitled "METHODS AND APPARATUS FOR MANAGING COLLECTIONS OF OBJECTS", which is incorporated herein by reference in its entirety.

Turning next to Fig. 7, an ORB daemon process 400 in accordance with a preferred embodiment of the present invention will now be described. ORB daemon process 400 is a background process executing on a computer system (such as computer system 30 of Fig. 2) which is part of a distributed network (such as network 10 of Fig. 1). For example, one suitable embodiment of ORB daemon process 400 may be a process such as daemon process_2 321 of Fig. 3. According to one aspect of the present invention, it is contemplated that each computer system within the distributed object op-

er identifier to the server process. A couple of method aspects of the present invention which utilize an object such as server process registration object 406 will be described in more detail below with reference to Figs. 11 - 14.

The method aspects of the present invention are discussed primarily from the viewpoint of the server and fall into substantially two categories. In the first category, methods for management (including fork and exec, and registration) of server processes are disclosed. A few preferred embodiments of the first case are described below in reference to Figs. 10, 12, and 13, each utilizing ORB daemon 400 of Fig. 7. In the second category, methods by which a server process starts (corresponding to the methods of Figs. 10, 12, and 13) are disclosed. A couple of preferred embodiments are described below with reference to Figs. 12 and 14. The need for the methods of the present invention often arise in response to a client call, even though the viewpoint of the methods is that of the server. Therefore, for the sake of clarity, the discussion of the methods of Figs. 10 - 14 will be preceded by a brief discussion of the client-server interaction from the viewpoint of the client.

Referring next to Fig. 9, a method 196 of invoking a target object in accordance with one embodiment of the present invention will be described. The target object invocation method begins in a step 198 when a client object running in a client process on the client host initiates the invocation of a target object. However, in the more general case, the client need not be an object. Typically, the client object will only have an indirection to the target object (*i.e.* an object reference) and will know the target object's interface requirements (or at least the portion of the interface requirements that apply to the information it is seeking). Therefore, the invocation begins when the client object calls the target object with a call that identifies the target object using the object reference and provides the arguments necessary to meet the target object's interface requirements. As illustrated in step 200, the client process responds to the client object's call with a call to the target object's surrogate which is located in the client process. Suitable methods for creating the target object surrogate and/or establishing a connection between the client object and the target object surrogate on the same machine and within the same process are known to those skilled in the art. In one example the object reference may be a pointer to the memory address of the surrogate. Typically a surrogate is created when arguments to an invocation are unmarshaled or when a reply from an invocation is unmarshaled. By way of example, a suitable method of creating the target object surrogate is described in Vanderbilt et. al.'s copending United States Patent Application Serial No. (Attorney Docket No. P721/SUN1P024) entitled "METHOD AND APPARATUS FOR DETERMINING THE TYPE OF AN OBJECT IN A DISTRIBUTED OBJECT SYSTEM" which is incorporated herein by reference in its entirety.

Once the call is received by the target object surrogate, the surrogate determines whether it already has addressing information for a target object servant existing on the server host. If the surrogate has the addressing information, then process control goes directly to a step 216 where it establishes a network connection with the server process if necessary (as will be described below in more detail). On the other hand, if the surrogate does not have addressing information for the target object servant, then this information must be obtained prior to establishing a connection. In the latter case, the target object surrogate proceeds to step 204 where it locates the server host. It should be appreciated that the server host will be identified in the object reference. Therefore, the target object surrogate may contact the ORB which will provide it with the addressing information for the ORB daemon process running on the server host.

Next, in a step 207, the target object surrogate establishes a network connection with the ORB daemon process on the server host. However, if the surrogate already has an established connection with the ORB daemon process, it may not be necessary to establish a second connection. In any event, once an appropriate connection is established, the target object surrogate requests, in a step 209, that the ORB daemon process return addressing information for the server process on which the target object servant is running. One suitable way of requesting this information is to call the locator object to invoke a look-up method. Of course, although steps 207 and 209 are logically explained as separate steps, they can be accomplished simultaneously in a single call. Methods of establishing connections between processes are well known in the art. One suitable method of establishing the aforementioned process is described in Brownell et. al.'s copending United States Patent Application Serial Number (attorney's docket no. P715/SUN1P018) entitled "METHOD AND APPARATUS FOR MANAGING CONNECTIONS FOR COMMUNICATION AMONG OBJECTS IN A DISTRIBUTED OBJECT SYSTEM" which is incorporated herein by reference in its entirety.

In response to steps 207 and 209, the ORB daemon must start and register the server process and then return addressing information to the client. A suitable method that the server host's ORB daemon process may go through will be described below with reference to Figs. 10 - 12. However, from the standpoint of the target object surrogate, it simply receives the requested server process addressing information in a step 214. With this knowledge in hand, the surrogate can establish a direct connection with the appropriate server process in step 216. Of course, it is plausible that the client process has previously established a connection with the server process for any variety of reasons such as a previous call to another target object running in this same server process. In this case, it may or may not be necessary (or desirable) to establish a second connection with the server process.

familiar to those of skill in the art. After the fork and exec of step 518, the locator object 404 proceeds to step 513 wherein it must wait until server process state entry 422 of active server table 402 transitions from starting to running. Once the server process is running, control proceeds to step 510 where addressing information is returned to the client.

In any event, once the addressing information is returned to the client in step 510, control proceeds to a step 512 where the operation look-up target object is complete. Additional embodiments of the method of Fig. 10 are envisioned. In particular, there may be additional states corresponding to the server process state 422 such as "unavailable" or "nonexistent." In either case, appropriate steps such as returning an error message to the client may be performed by locator object 404. Another state may be "forward all requests." In this case, the locator object 404 may return addressing information for a "replacement" target object and/or server process.

Other variations on the method of Fig. 10 are envisioned. During normal operation of a distributed object operating environment there may arise situations wherein the target object and/or server process may be temporarily (or permanently) unavailable. For example, maintenance operations such as backing up, fixing code, or installing new objects may disable the target object and/or server process. According to one embodiment of the present invention, there may be three possible states: long hold down, short hold down, and no hold down. In the case of no hold down, both the target object and the server process are available and the locator object 404 thread of execution may proceed exactly as described in reference to Fig. 10. In the case of short hold down, the thread of execution is blocked temporarily, similar to step 513 of Fig. 10. This case would be appropriate if the delay was relatively short. In the case of long hold down, an error message would be returned to the client. This would be appropriate if the target object and/or server process was permanently unavailable, or unavailable for a relatively long period of time. One suitable place to store the hold down information is within the execdef. Then, step 504 of Fig. 10 could be expanded to further include the step of determining the hold down and performing the appropriate steps.

Referring next to Fig. 11, a method of starting a server process in accordance with one embodiment of the present invention will now be described. By way of example, the server process may start in a step 550 in response to the fork and exec step 518 of Fig. 10. Next, in a step 552, the server process calls a transport service to create a communications port and form its addressing information for the server process. As will be appreciated by those skilled in the art, the transport service may be implemented as part of the host computer operating system or as part of the distributed object operating environment. Then, in a step 554, the server process calls the server process registration object 406 to invoke the register server operation, passing server identifier 420, addressing information 424, and server process identification 426 as arguments. With this information the server process registration object 406 can uniquely register the server process as described in further detail below with respect to Fig. 12, thereby perpetuating the advantages of the present invention. Once this call is made and the server process is registered, the server process is ready to handle requests in a step 556.

Turning now to Fig. 12, a method of registering a server process in an active server table 402 in accordance with one embodiment of the present invention will be described. By way of explanation, a server process registration object may perform the registration in response to an invocation as in step 554 of Fig. 11. In a step 570, server process registration object 406 receives a register server invocation which includes arguments such as server identifier 420, addressing information 424, and server process identification 426. Next, in a step 572, server process registration object 406 stores addressing information 424 in active server table 402. Note that an entry for this server process (identifying it by server identifier 420) was created in a step such as step 514 of Fig. 10. Then, in a step 574, server process registration object 406 marks server process state 422 as running. Once the state is marked running, server process registration object 406 has completed the requested service and is done in a step 576.

As will be appreciated, the methods of Figs. 9 - 12 may all be woven into one client-server interaction encompassing the steps 356 - 362 as described in reference to the client-server paradigm of Fig. 6. Each of the Figs. 9 - 12 may be interpreted to describe a separate thread of execution, and one embodiment of the present invention may be implemented this way. Additionally, these threads may be collapsed into a single thread for implementation on a single-threaded system. The advantages of different threading schemes, as well as their implementation and construction, will be well familiar to those skilled in the art. Alternatively, various embodiments would provide methods for the different client-server interactions described above with reference to Figs. 3 - 5. In light of the preceding it will be apparent to those skilled in the art how to construct and implement these and still further embodiments, all which fall within the scope of the present invention.

Underlying the above discussion of Figs. 9 - 12 was the assumption that an ORB daemon process 400 started a server process. However, this is not always the case. For instance, a client may pass a call directly to the target object yet the target object and/or the server process may not be active. In another instance, a distributed object and/or its corresponding server process may wish to activate due to one of many other causes (*e.g.* a particular object and/or process always executes at power up). In either instance, a mechanism is required for server processes to "self-start" (*i.e.* start-up

ject which could be termed the locator service. Additionally, other process for execution control (besides threads) may be used to implement the present invention.

Therefore, the present examples are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope of the appended claims.

Claims

1. A daemon process for use on a computer system in a distributed object operating environment, said distributed object operating environment including a plurality of distributed objects which are intended to reside in a corresponding plurality of server processes, said daemon process operable to manage that portion of said plurality of distributed objects which reside on said computer system and that portion of said plurality of server processes which execute on said computer system, said daemon process including:

    an active server table arranged to maintain entries regarding a plurality of server processes, said entries including a server identifier, a server process state, and server process addressing information; and
    a locator service operable to access said active server table to provide server process addressing information for said plurality of server processes, said locator service further operable to register a server process in said active server table.

2. A daemon process as described in claim 1 further including an object adapter database, said object adapter database including:

    a plurality of target object identifiers;
    a plurality of server identifiers, each server identifier corresponding to at least one of said plurality of target object identifiers; and
    a plurality of object references, each object reference corresponding to one of said plurality of target object identifiers.

3. A daemon process as described in claim 1 wherein said locator service includes a locator object arranged to perform said operation of accessing said active server table to provide server process addressing information, said operation performed in response to a request from a client.

4. A daemon process as described in claim 1 wherein said locator service includes a server process registration object arranged to perform said operation of registering a server process in the active server table.

5. A distributed object operating environment comprising:

    a plurality of computer systems;
    a computer network interconnecting said plurality of computer systems; and
    at least one daemon process as recited in claim 1.

6. A computer system for use in a distributed object operating environment, said computer system comprising:

    a central processing unit;
    a memory accessible by said central processing unit, said memory including a plurality of distributed objects; and
    a locator service implemented on said computer system that manages said plurality of distributed objects and a plurality of server processes which may be implemented on said computer system.

7. A computer system as recited in claim 6 further including an input/output device which may communicate with said distributed object operating environment and wherein said locator service includes:

    a computer implemented receiver for receiving a look-up call for a target object from a client, said look-up call received via said input/output device;
    a computer implemented mechanism for obtaining a server identifier for said target object;
    a computer implemented evaluator for determining a state of a server process corresponding to said target object, said server process uniquely defined within said computer system by said server identifier, said state of server process being one of the group consisting of running, starting, and not active; and
    a computer implemented transmitter for returning addressing information corresponding to said server process to said client, said addressing information returned via said input/output device.

8. A computer system as recited in claim 7 wherein said transmitter is arranged to operate immediately upon said evaluator determining that said state is running.

9. A computer system as recited in claim 7 further including a wait device operable to wait until said state transitions from starting to running.

tem for use in a distributed object operating environment, said method performed by a daemon process resident on said computer system, said method comprising the steps of:

receiving under computer control a look-up call for a target object, said look-up call originating from a client;
obtaining under computer control a server identifier for said target object;
determining under computer control a state of a server process corresponding to said target object, said server process uniquely defined within said computer system by said server identifier; and
returning under computer control addressing information corresponding to said server process to said client.

21. A method as recited in claim 20 wherein said state is determined to be running and therefore said step of returning addressing information is performed immediately.

22. A method as recited in claim 20 wherein said state is determined to be starting and therefore said method further includes a step of waiting under computer control until said state transitions from starting to running, said waiting step performed prior to the step of returning addressing information.

23. A method as recited in claim 20 wherein the step of obtaining a server identifier for said target object includes accessing under computer control a first database.

24. A method as recited in claim 23 wherein said first database is an object adapter database.

25. A method as recited in claim 23 wherein the step of determining a state of a server process includes looking up under computer control said state in a second database.

26. A method as recited in claim 25 wherein said second database is a active server table.

27. A method as recited in claim 25 wherein said first and second database are the same database.

28. A method as recited in claim 25 wherein said server process is not listed in said second database and therefore the method further includes the following steps performed prior to said return addressing information step:

creating under computer control a server process entry in said second database, said server process entry including a server identifier and a server process state;
marking under computer control said server process state as starting in said second database;
accessing under computer control said first database to get an execdef for said target object;
starting under computer control said server process using said execdef for target object; and
waiting under computer control until said server process state for said server process transitions from starting to running.

29. A method as recited in claim 28 wherein said server process responds to said step of starting said server process by performing the steps of:

receiving under computer control a server process identification number from an operating system of said computer system;
creating under computer control a communications port for said server process;
forming under computer control addressing information for said server process;
calling under computer control a server process registration object resident in said daemon process, said call operable to invoke a register server operation, said call passing said addressing information, said server process identification number, and said server identifier to said server process registration object.

30. A method as recited in claim 29 wherein said server process registration object performs the steps of:

receiving under computer control a call invoking said register server operation;
storing under computer control said addressing information in said second database; and
marking under computer control said server process state entry as running.

31. A computer implemented method for a server process executing on a computer system, said server process having a server identifier, said computer system for use in a distributed object operating environment, said method comprising the steps of:

receiving under computer control a server process identification number from an operating system of said host computer;
creating under computer control a communications port for said server process;
forming under computer control addressing information for said server process;
calling under computer control a server process registration object resident in a process execut-

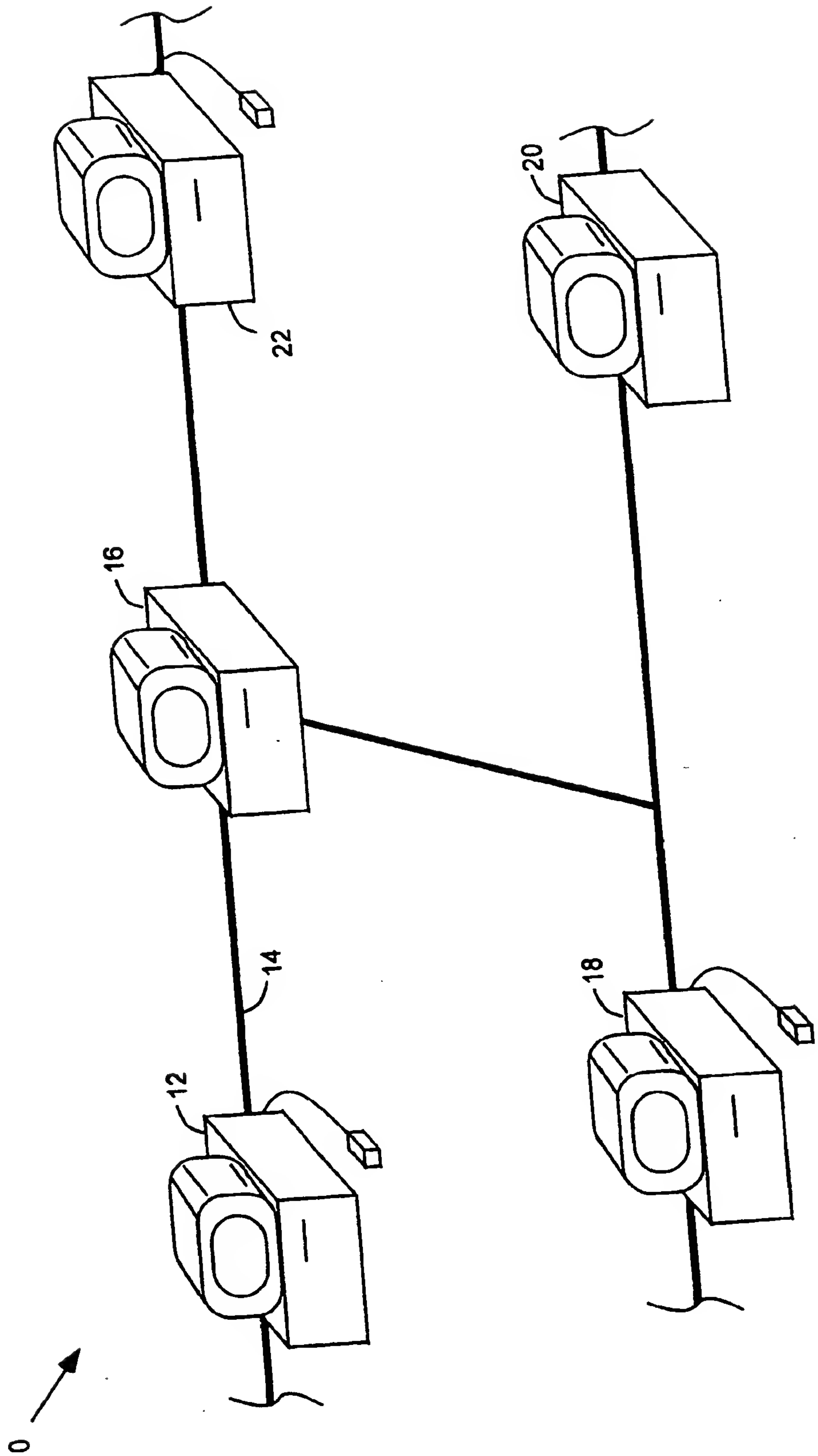

Fig. 1

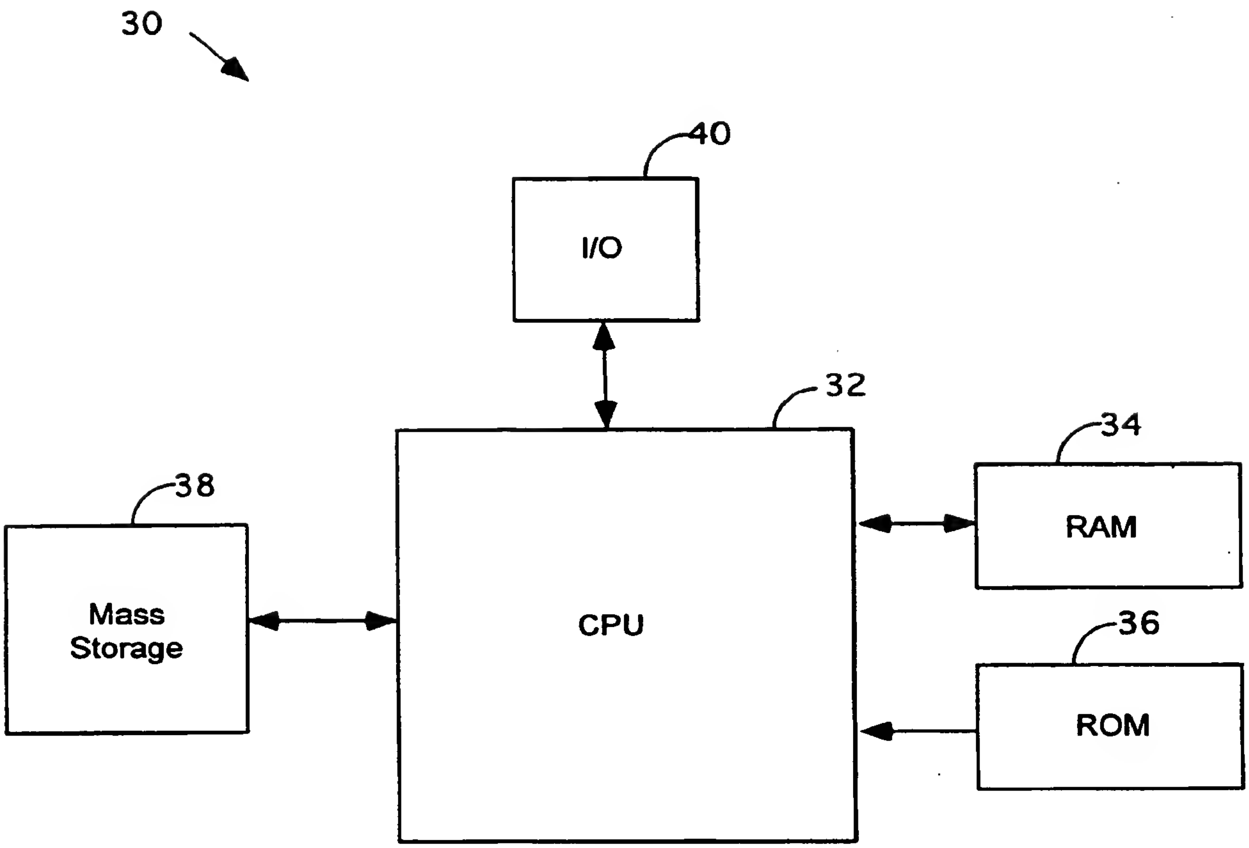

Fig. 2

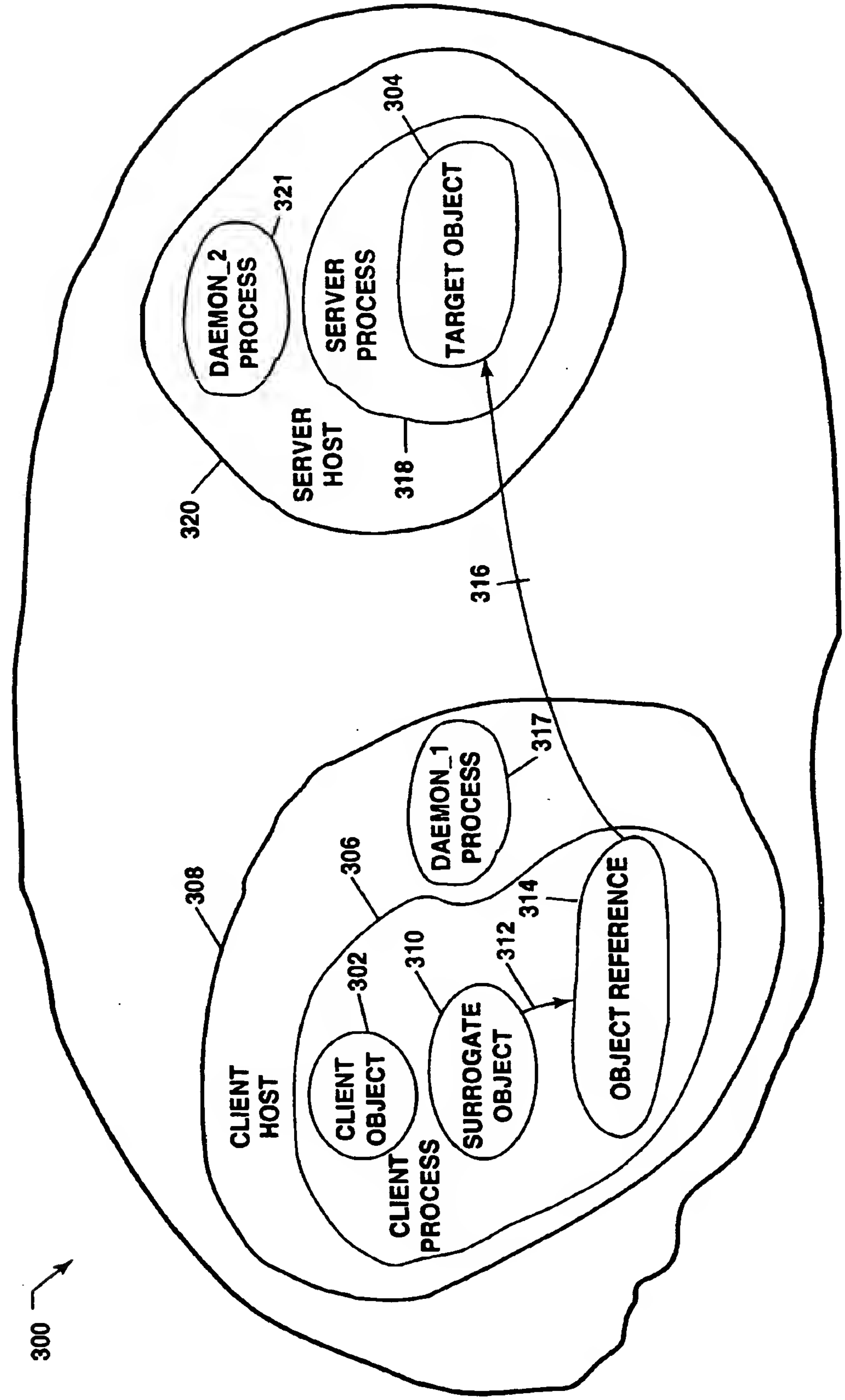
DAEMON_2 PROCESS
321
304
SERVER PROCESS
TARGET OBJECT
SERVER HOST
318
320
316
DAEMON_1 PROCESS
317
308
306
314
OBJECT REFERENCE
312
310
302
CLIENT HOST
CLIENT OBJECT
SURROGATE OBJECT
CLIENT PROCESS
300

Fig. 3

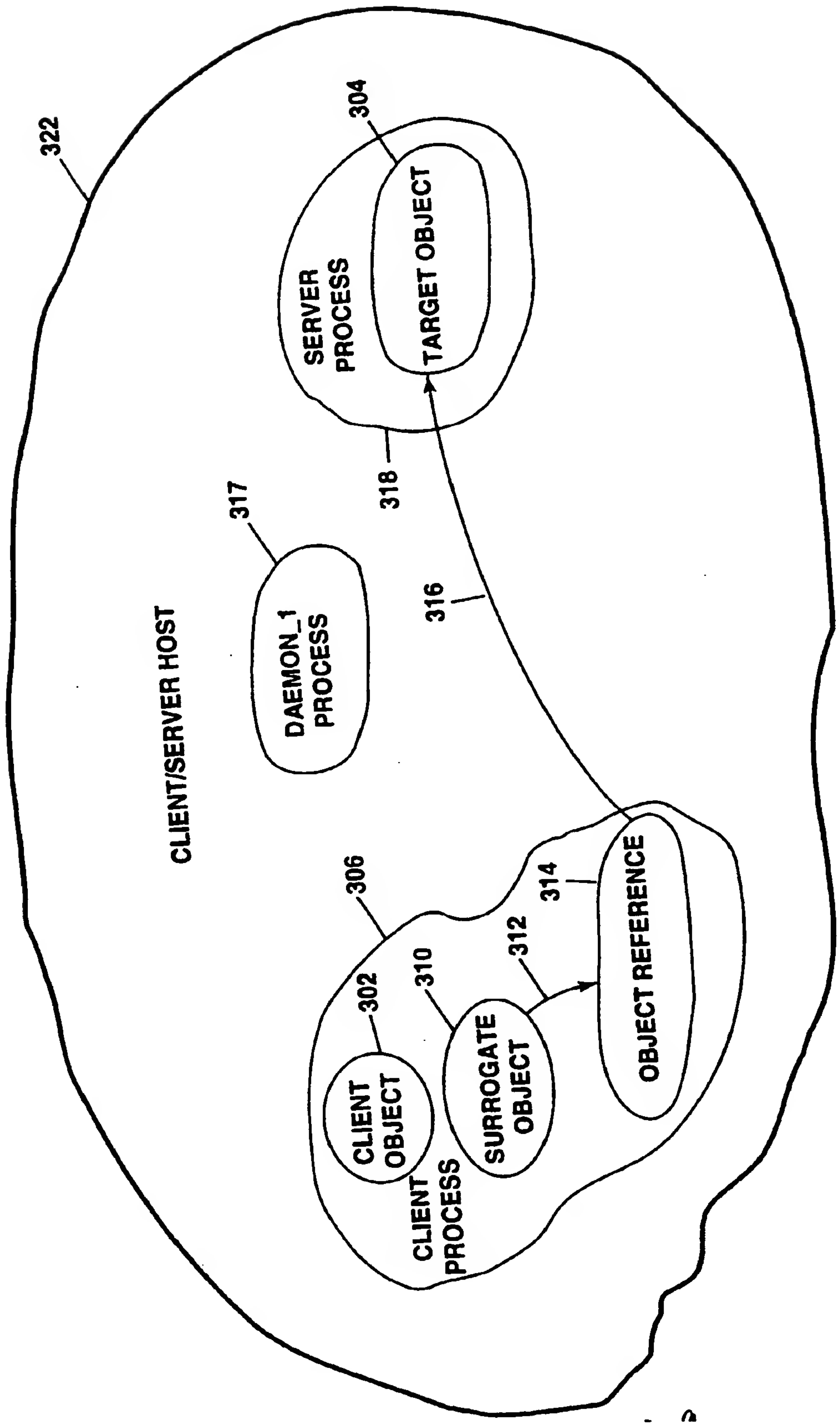
322
304
SERVER PROCESS
TARGET OBJECT
318
317
DAEMON_1 PROCESS
316
CLIENT/SERVER HOST
306
314
OBJECT REFERENCE
312
310
302
CLIENT OBJECT
SURROGATE OBJECT
CLIENT PROCESS

Fig. 4

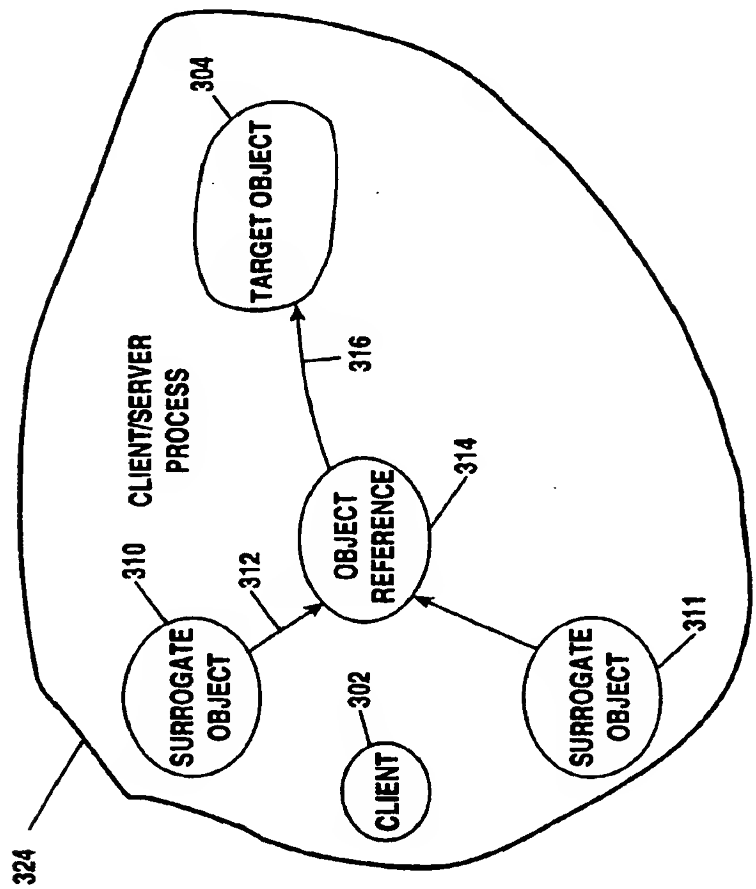
324
CLIENT/SERVER PROCESS
SURROGATE OBJECT
310
312
OBJECT REFERENCE
314
316
TARGET OBJECT
304
CLIENT
302
SURROGATE OBJECT
311

Fig. 5

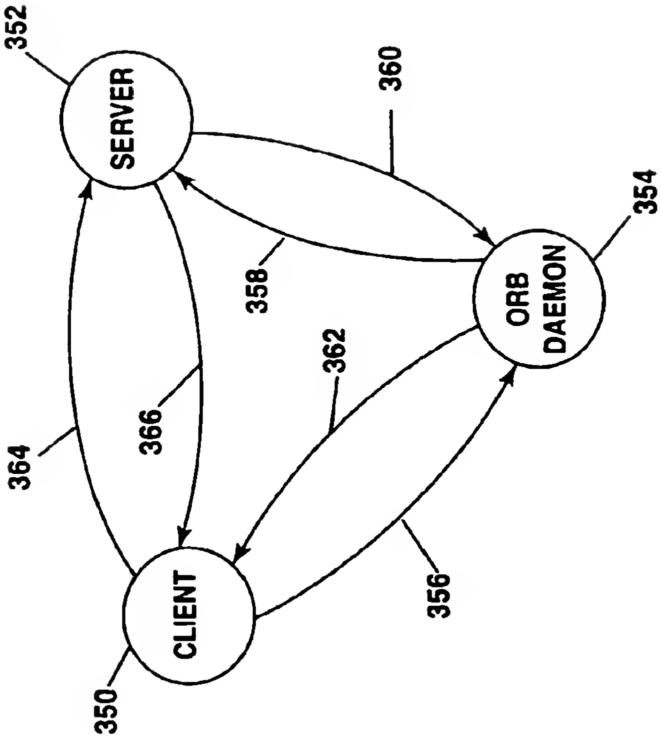
SERVER
352
360
358
ORB DAEMON
354
362
366
364
CLIENT
350
356

Fig. 6

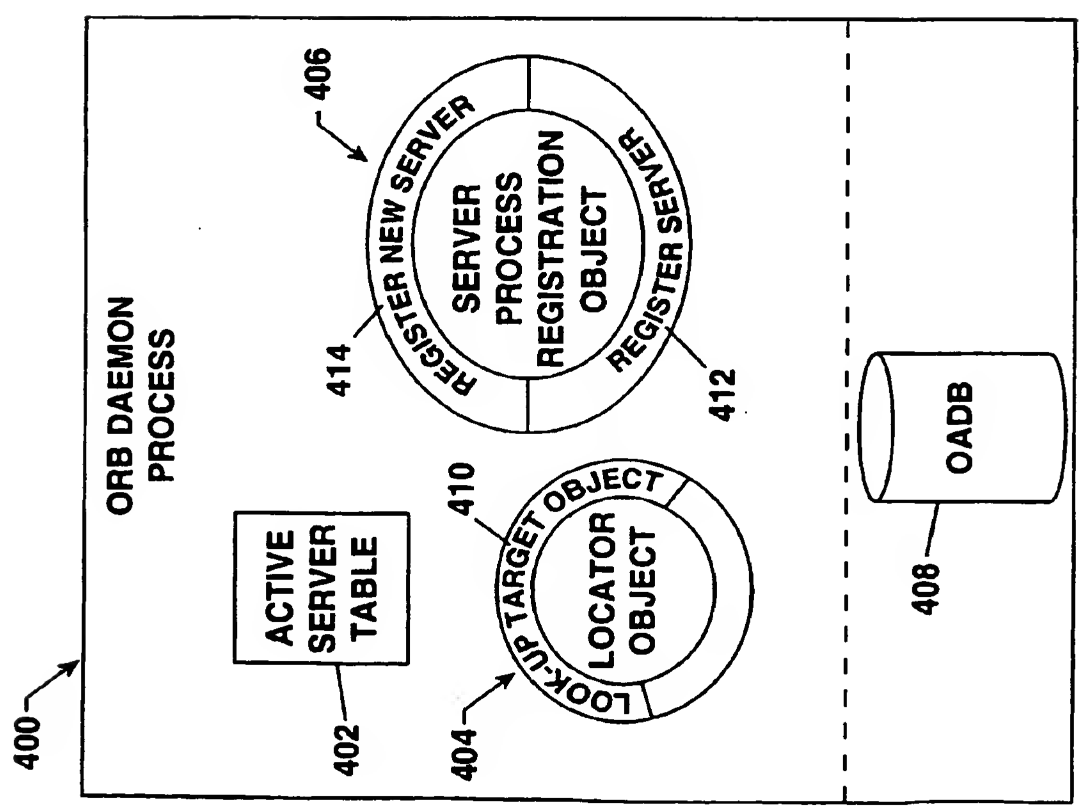
400
ORB DAEMON PROCESS
ACTIVE SERVER TABLE
402
LOOK-UP TARGET OBJECT
LOCATOR OBJECT
404
410
406
REGISTER NEW SERVER
SERVER PROCESS REGISTRATION OBJECT
REGISTER SERVER
414
412
OADB
408

Fig. 7

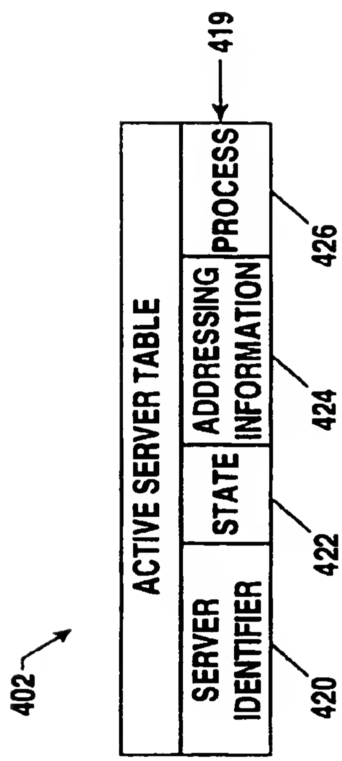
402
ACTIVE SERVER TABLE
SERVER IDENTIFIER
STATE
ADDRESSING INFORMATION
PROCESS
419
420
422
424
426

Fig. 8

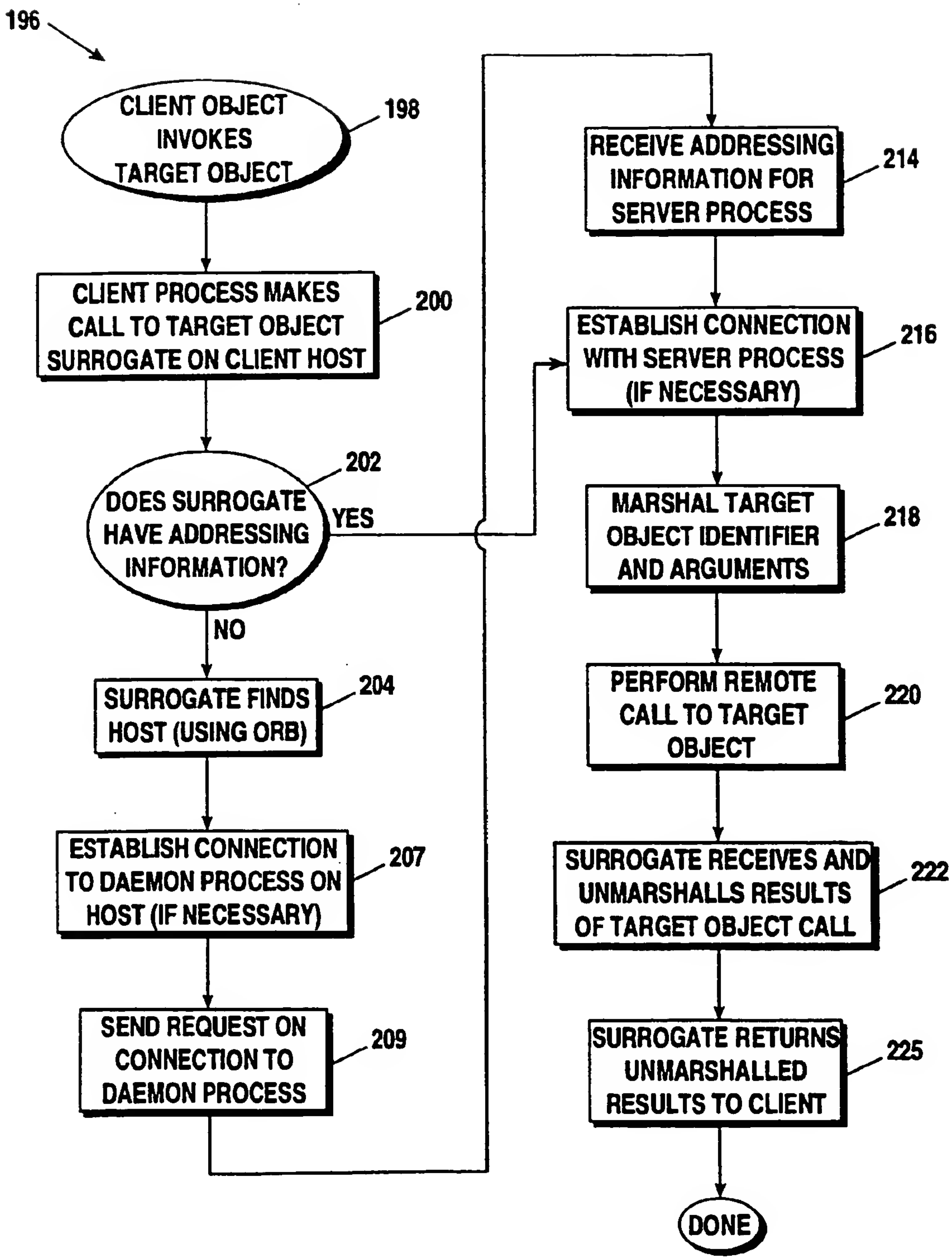
196
CLIENT OBJECT INVOKES TARGET OBJECT
198
CLIENT PROCESS MAKES CALL TO TARGET OBJECT SURROGATE ON CLIENT HOST
200
DOES SURROGATE HAVE ADDRESSING INFORMATION?
202
YES
NO
SURROGATE FINDS HOST (USING ORB)
204
ESTABLISH CONNECTION TO DAEMON PROCESS ON HOST (IF NECESSARY)
207
SEND REQUEST ON CONNECTION TO DAEMON PROCESS
209
RECEIVE ADDRESSING INFORMATION FOR SERVER PROCESS
214
ESTABLISH CONNECTION WITH SERVER PROCESS (IF NECESSARY)
216
MARSHAL TARGET OBJECT IDENTIFIER AND ARGUMENTS
218
PERFORM REMOTE CALL TO TARGET OBJECT
220
SURROGATE RECEIVES AND UNMARSHALLS RESULTS OF TARGET OBJECT CALL
222
SURROGATE RETURNS UNMARSHALLED RESULTS TO CLIENT
225
DONE

Fig. 9

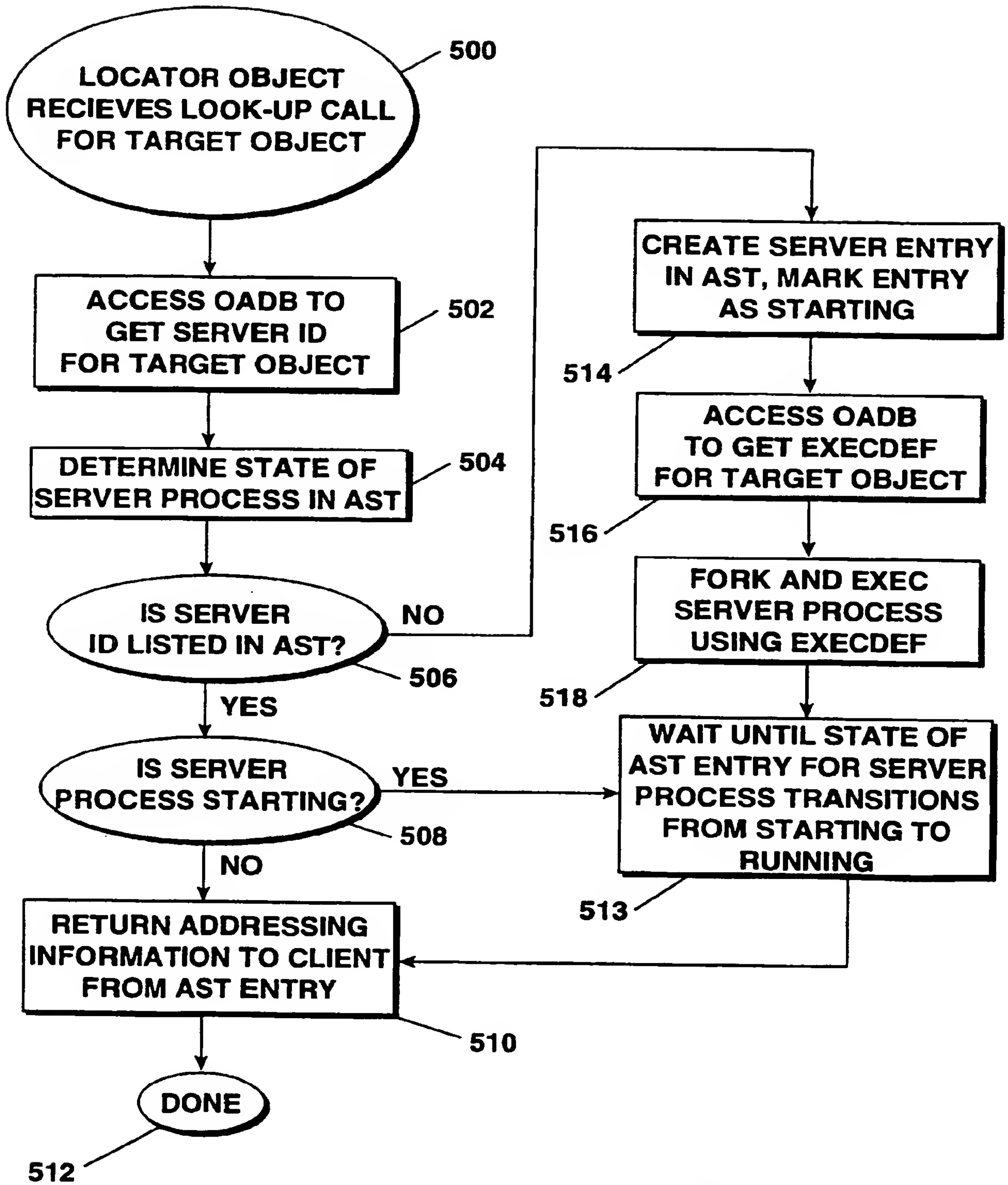
LOCATOR OBJECT RECIEVES LOOK-UP CALL FOR TARGET OBJECT
500
ACCESS OADB TO GET SERVER ID FOR TARGET OBJECT
502
DETERMINE STATE OF SERVER PROCESS IN AST
504
IS SERVER ID LISTED IN AST?
NO
506
YES
IS SERVER PROCESS STARTING?
YES
508
NO
RETURN ADDRESSING INFORMATION TO CLIENT FROM AST ENTRY
510
DONE
512
CREATE SERVER ENTRY IN AST, MARK ENTRY AS STARTING
514
ACCESS OADB TO GET EXECDEF FOR TARGET OBJECT
516
FORK AND EXEC SERVER PROCESS USING EXECDEF
518
WAIT UNTIL STATE OF AST ENTRY FOR SERVER PROCESS TRANSITIONS FROM STARTING TO RUNNING
513

Fig. 10

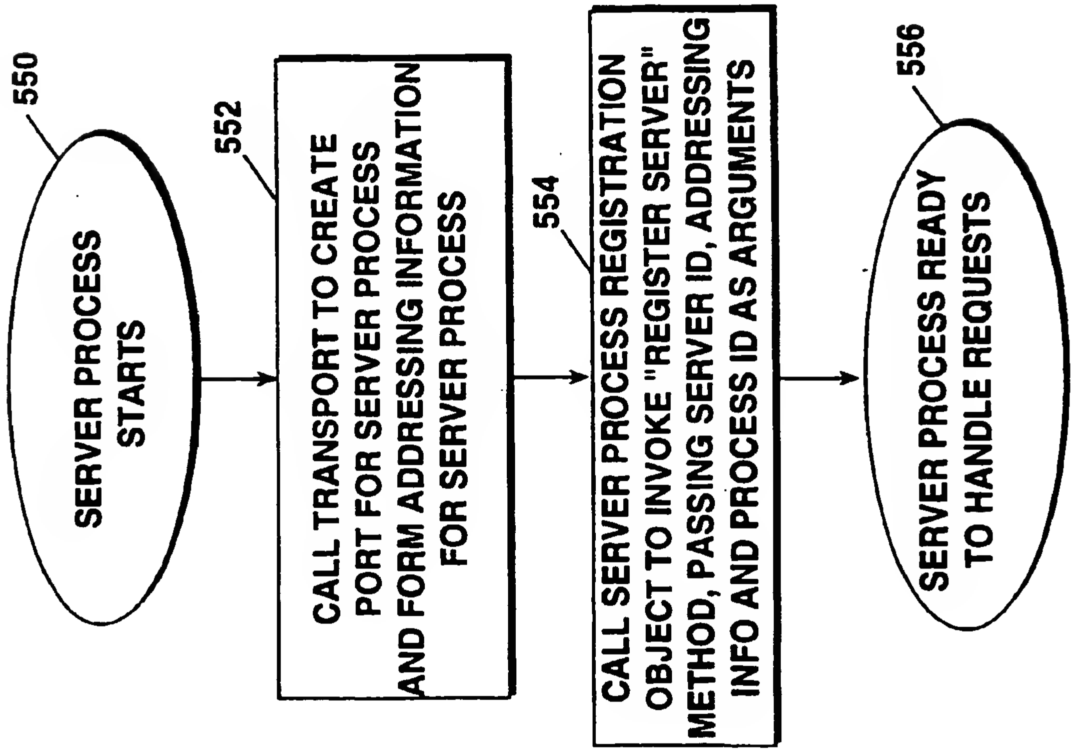
550
SERVER PROCESS STARTS
552
CALL TRANSPORT TO CREATE PORT FOR SERVER PROCESS AND FORM ADDRESSING INFORMATION FOR SERVER PROCESS
554
CALL SERVER PROCESS REGISTRATION OBJECT TO INVOKE "REGISTER SERVER" METHOD, PASSING SERVER ID, ADDRESSING INFO AND PROCESS ID AS ARGUMENTS
556
SERVER PROCESS READY TO HANDLE REQUESTS

Fig. 11

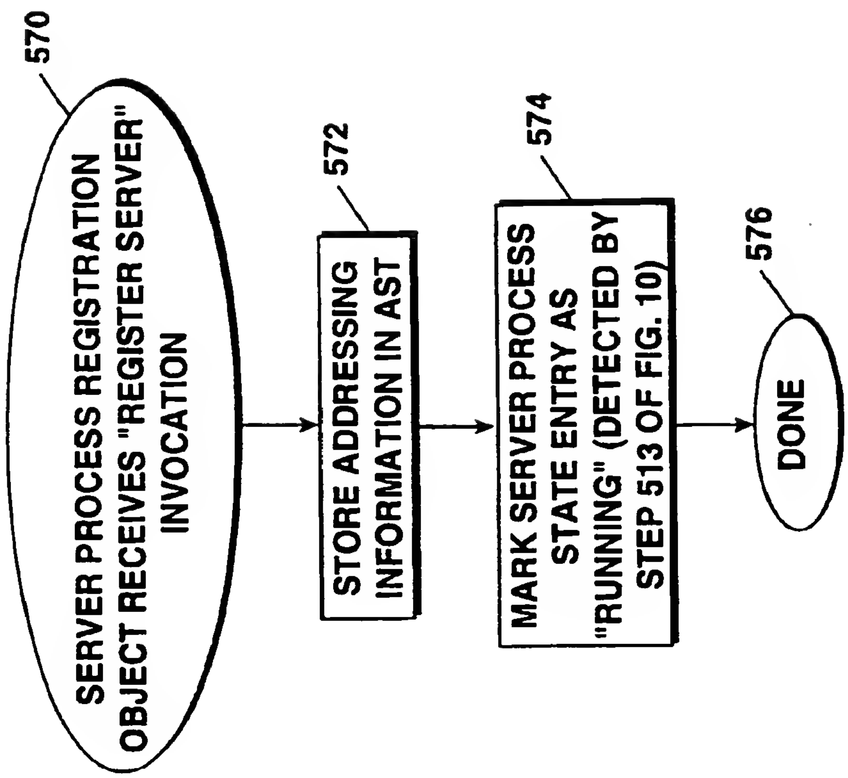
570
SERVER PROCESS REGISTRATION OBJECT RECEIVES "REGISTER SERVER" INVOCATION
572
STORE ADDRESSING INFORMATION IN AST
574
MARK SERVER PROCESS STATE ENTRY AS "RUNNING" (DETECTED BY STEP 513 OF FIG. 10)
576
DONE

Fig. 12

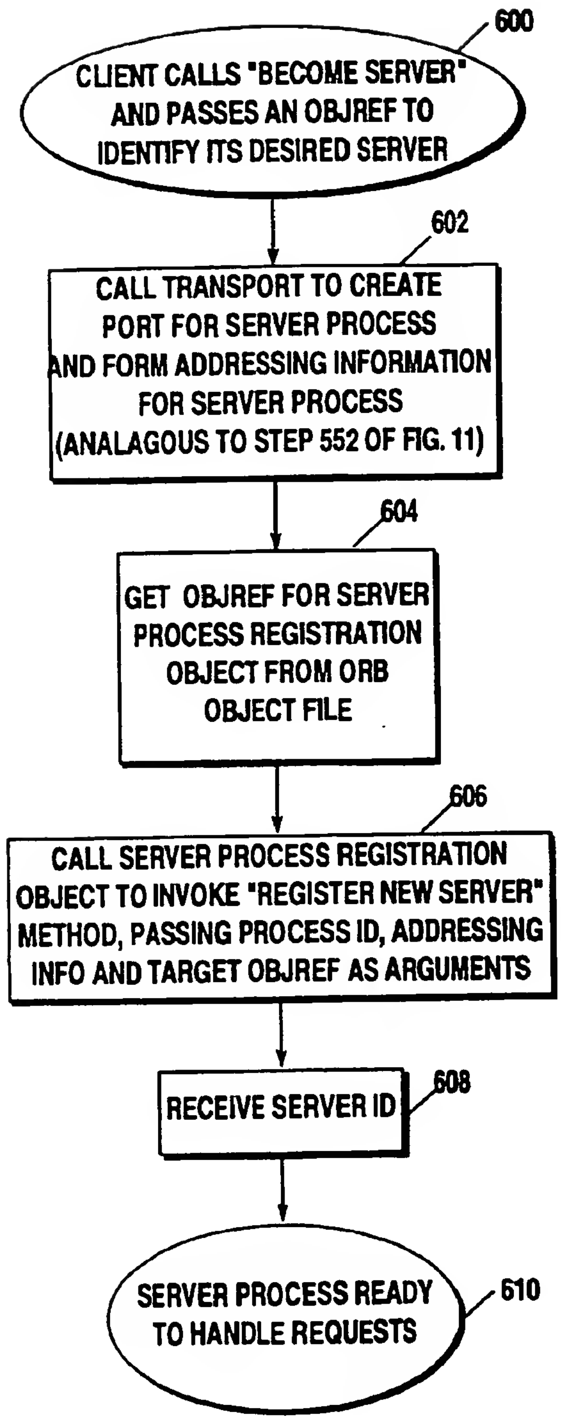
600
CLIENT CALLS "BECOME SERVER" AND PASSES AN OBJREF TO IDENTIFY ITS DESIRED SERVER
602
CALL TRANSPORT TO CREATE PORT FOR SERVER PROCESS AND FORM ADDRESSING INFORMATION FOR SERVER PROCESS (ANALAGOUS TO STEP 552 OF FIG. 11)
604
GET OBJREF FOR SERVER PROCESS REGISTRATION OBJECT FROM ORB OBJECT FILE
606
CALL SERVER PROCESS REGISTRATION OBJECT TO INVOKE "REGISTER NEW SERVER" METHOD, PASSING PROCESS ID, ADDRESSING INFO AND TARGET OBJREF AS ARGUMENTS
608
RECEIVE SERVER ID
610
SERVER PROCESS READY TO HANDLE REQUESTS

Fig. 13

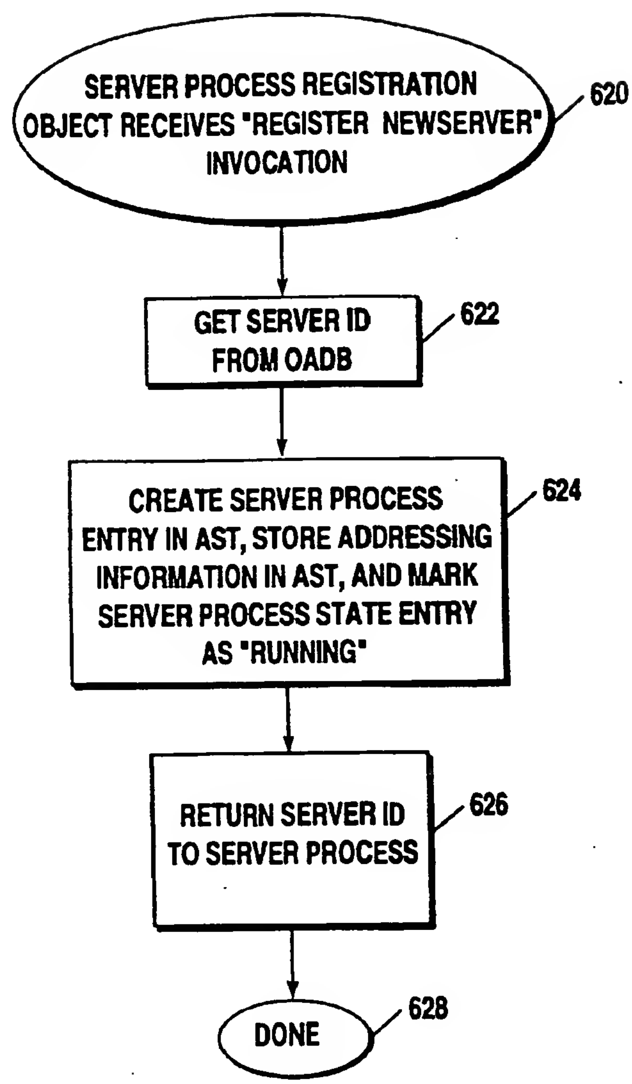
620
SERVER PROCESS REGISTRATION OBJECT RECEIVES "REGISTER NEWSERVER" INVOCATION
622
GET SERVER ID FROM OADB
624
CREATE SERVER PROCESS ENTRY IN AST, STORE ADDRESSING INFORMATION IN AST, AND MARK SERVER PROCESS STATE ENTRY AS "RUNNING"
626
RETURN SERVER ID TO SERVER PROCESS
628
DONE

Fig. 14

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 96 30 1560

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	PROCEEDINGS SECOND INTERNATIONAL WORKSHOP OBJECT ORIENTATION IN OPERATING SYSTEMS, 24-25 SEP. 1992, DOURDAN, FR, 1 January 1992, pages 212-220, XP002004311 DAVE A ET AL: "PROXIES, APPLICATION INTERFACES, AND DISTRIBUTED SYSTEMS" * page 213, left-hand column, line 45 - right-hand column, line 1 * * page 214, left-hand column, line 8 - page 218, left-hand column, line 20 * ---	1-34	G06F9/46
A	C++ REPORT, MARCH-APRIL 1994, USA, vol. 6, no. 3, ISSN 1040-6042, pages 50-59, XP000577570 SCHMIDT D: "A domain analysis of network daemon design dimensions" * page 51, right-hand column, line 18 - page 52, left-hand column, line 9 * * page 58, left-hand column, line 45 - page 59, right-hand column, line 3 * -----	1-34	
			TECHNICAL FIELDS SEARCHED (Int.Cl.6)
			G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	26 July 1996	Brandt, J

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)